Problem 5 (5 points)

In [38]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.linear_model import LogisticRegression
```

One-vs-All (One-vs-Rest) Multinomial Classification

Load Dataset

(Don't edit this)

- (x,y) values are stored in rows of `xy`
- class values are in `c`

In [39]:
```
x = np.array([7.4881350392732475,16.351893663724194,22.427633760716436,29.048831
y = np.array([0.11120957227224215,0.1116933996874757,0.14437480785146242,0.11818
xy = np.vstack([x,y]).T
c = np.array([0,2,2,2,2,2,0,2,2,2,2,2,0,0,2,0,1,2,0,0,1,1,1,2,0,1,0,1,1,1,0,0,1,
```

Binomial classification function

You are given a function that performs binomial classification by using sklearn's `LogisticRegression` tool: `prob = get_ovr_prob_function(xy, c, A)`

To use it, input:

- `xy` , an array in which each row contains (x,y) coordinates of data points
- `c` , an array that specifies the class each point in `xy` belongs to
- `A` , the class of the group (0, 1, or 2 in this problem) -- classifies into A or "rest"

The function outputs a probability function (`prob()` in this case), used to determine the probability that each `xy` is class A or [not A], such as by using `prob(xy)` .

In [40]:
```
def get_ovr_prob_function(xy, c, A):
    c_new = (c == A).astype(int)

    model = LogisticRegression()
    model.fit(xy,c_new)

    def prob(xy):
        pred = model.predict_proba(xy)[:,1]
        return pred.flatten()

    return prob
```

Coding an OvR classifier

Now you will create a one-vs-rest classifier to do multinomial classification. Binomial predictions will be made for each class vs. the rest of the classes. The class whose binomial prediction gives the highest probability is the selected class.

Complete the two functions we have started:

- `generate_ovr_prob_functions(xy, c)` which returns a list of binary classifier probability functions for all possible classes (0, 1, and 2 in this problem)
- `classify_ovr(probs, xy)` which loops through a list of ovr classifier probabilities and gets the probability of belonging to each class, for each point in `xy`. Then taking the highest probability for each, return the overall class predictions for each point.

In [57]:
```
def generate_ovr_prob_functions(xy, c):
    # YOUR CODE GOES HERE
    prob_functions = []
    for i in range(0,3):
        prob = get_ovr_prob_function(xy,c,i)
        prob_functions.append(prob)
    return prob_functions

def classify_ovr(probs, xy):
    # YOUR CODE GOES HERE
    pred = []
    overall_pred = []

    for i in probs:
        pred.append(i(xy))

    pred = np.vstack(pred).T

    for i in range(xy.shape[0]):
        overall_pred.append(np.argmax(pred[i,:]))

    overall_pred = np.array(overall_pred)
    return overall_pred
```

Trying out our multinomial classifier:

In [58]:
```
probs = generate_ovr_prob_functions(xy, c)
preds = classify_ovr(probs, xy)
accuracy = np.sum(preds == c) / len(c) * 100
print("True Classes:", c)
print(" Predictions:", preds)
print("    Accuracy:", accuracy, r"%")
```

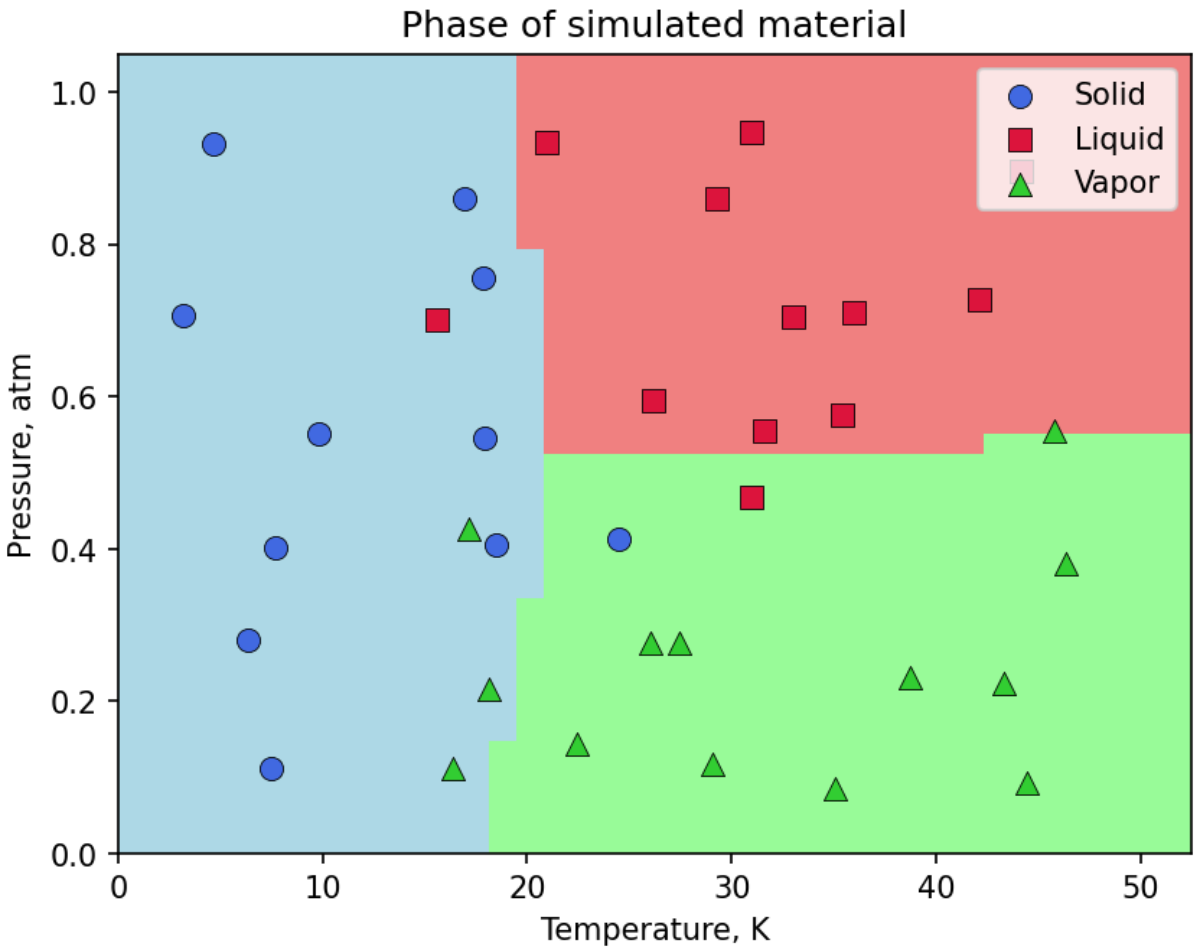
```
True Classes: [0 2 2 2 2 2 0 2 2 2 2 2 0 0 2 0 1 2 0 0 1 1 1 2 0 1 0 1 1 1 0 0
1 1 1 1]
 Predictions: [0 0 2 2 2 2 0 0 2 2 2 2 0 0 0 2 2 2 0 0 1 1 1 1 0 0 0 1 1 1 0 0
1 1 1 1]
    Accuracy: 80.55555555555556 %
```

Plotting multnomial classifier results

Here, we have made some plotting functions -- run these cells to visualize the decision boundaries.

In [59]:
```
def plot_data(x, y, c, title="Phase of simulated material", newfig=True):
    xlim = [0,52.5]
    ylim = [0,1.05]
    markers = [dict(marker="o", color="royalblue"), dict(marker="s", color="crims
    labels = ["Solid", "Liquid", "Vapor"]

    if newfig:
        plt.figure(dpi=150)

    for i in range(1+max(c)):
        plt.scatter(x[c==i], y[c==i], s=60, **(markers[i]), edgecolor="black", li

    plt.title(title)
    plt.legend(loc="upper right")
    plt.xlim(xlim)
    plt.ylim(ylim)
    plt.xlabel("Temperature, K")
    plt.ylabel("Pressure, atm")
    plt.box(True)

def plot_ovr_colors(probs, res=40):
    xlim = [0,52.5]
    ylim = [0,1.05]
    xvals = np.linspace(*xlim,res)
    yvals = np.linspace(*ylim,res)
    x,y = np.meshgrid(xvals,yvals)
    XY = np.concatenate((x.reshape(-1,1),y.reshape(-1,1)),axis=1)

    color = classify_ovr(probs,XY).reshape(res,res)

    cmap = ListedColormap(["lightblue","lightcoral","palegreen"])
    plt.pcolor(x, y, color, shading="nearest", zorder=-1, cmap=cmap,vmin=0,vmax=2
    return
```

In [60]:
```
plot_data(x,y,c)
plot_ovr_colors(probs)
plt.show()
```



In [ ]:

In [ ]: